DNN Complier的基础认知：

输入输出

DNNC 接受高级DNN模型规范作为输入，并为不同的硬件架构（如CPU、GPU和各种硬件加速器）生成优化的DNN可执行文件。
DL编译器主要目的是将深度学习框架描述的模型在各种硬件平台上生成有效的代码实现，其完成的模型定义到特定代码实现的转换将针对模型规范和硬件体系结构高度优化

编译过程：

编译的过程需要将中间高级模型转化成几个中间过渡模型并且还要实现依赖平台或者不依赖平台的对基于规则或者基于学习的优化。

关于IR

编译器前端和编译器后端，中间表示（IR）分布在前端和后端。通常，IR是程序的抽象，用于程序优化。
传统编译器里分为前端，优化和后端，其中前端和语言打交道，后端和机器打交道，现代编译器的的前端和后端分的很开，而他们之间的共同桥梁就是IR。IR可以说是一种胶水语言，注重逻辑而去掉了平台相关的各种特性，这样为了支持一种新语言或新硬件都会非常方便。

值得一说的是，dnn的编译器到现在都还没有系统而且规范的理解，更不用说编译器判断标准.

编译器的前端：图IR以及优化

将深度神经网络的计算图给转化为图IR,就是编译器的前端工作
将编译过程的知识/信息保存在图中，对应的算法通过对图中的对象（节点、边、列表和树）操作来描述。

后端：

低级IR和优化

低级IR是从图IR生成的，用于特定于硬件的优化。低级IR通常涉及与内存相关的操作

调度和调优

将DNN运算符映射到低级代码的策略称为调度，即发现参数设置

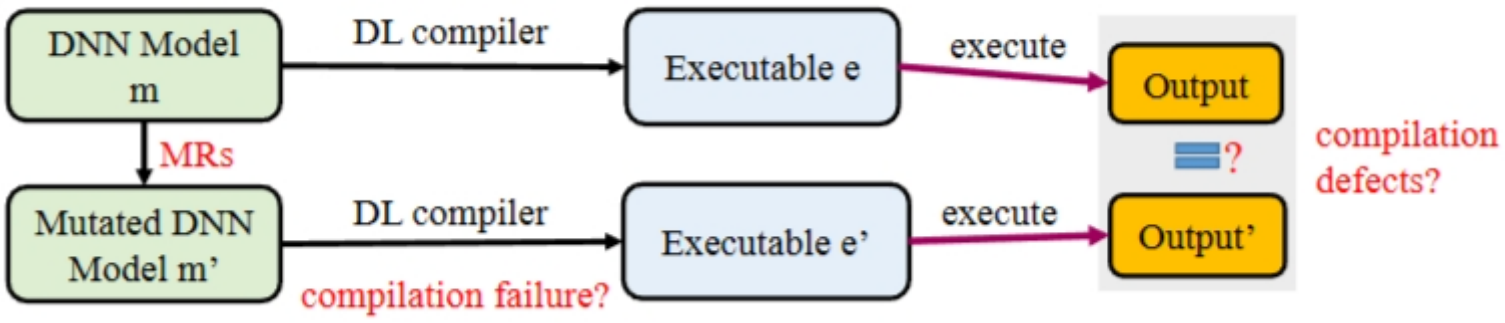
代码生成和优化

低级IR进一步编译

MT-DLComp 框架

这个框架是利用了变质测试来检测编译器的编译错误.而且这个框架是由深度神经网络模型经过特定的蜕变关系产生另一个语义相同的DNN模型。（而不是由输入变质）
再用相同标签的输入来观察两个模型的输出之间的误差。

$E\ D\ (m).Label = E\ D\ (m').Label$ (1)
$|E\ D\ (m).Score - E\ D\ (m').Score| < \delta$



蜕变测试：

蜕变测试就是说对比由蜕变关系得到的测试用例在程序的输出与原测试案例在程序的输出，
若出现了不一致的输出则代表程序存在缺陷.
是用来检查程序的多个执行结果之间的关系来测试程序，有效解决待测程序的预期输出难以确定（从而难以测试）的问题。

预言机3：

论文中提及的预言机3其实就是一个能找出两个语义相同的DNN模型之间的不一致性的抽象计算机

框架搭建的方法论

1.将seed DNN模型经过两次蜕变函数的迭代得到一个蜕变的模型集合m'
2.UOC(通用不透明条件)，PIOC(输入不透明条件),这两个就是该实验使用的蜕变关系
这两种方案共同用于迭代变异并最大限度地提高在DL编译器中触发输出不一致的机会

Universal Obscure Condition (UOC) ：

通过引入一个DNN运算符序列，这些运算符在运行时总是产生零作为最终输出。这种方案被称为通用模糊条件（UOC），因为尽管模型输入值不同，它总是产生零。如下：

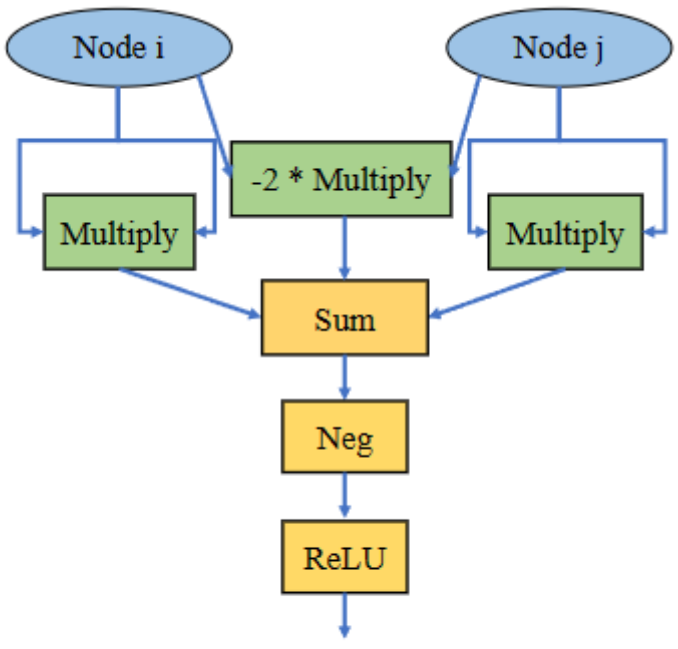


Fig. 5. Simplified view of UOC.

i，j分别是结点i和结点j的输出，这里构造了一个数论结构：
$i2 - 2 \times i \times j + j2 \geq 0$;在获得求和之后再通过Neg结点让结果取负，
最后再ReLU结点上调用一个激活函数$f\ (x) = max\ (0, x)$
所以此DNN运算符序列的最终输出是：ReLU(−1 × $(i - j)2$)
此时无论两个输入结点的输出值是什么，最终结果都会是0

更重要的是，我们发现DL编译器通常无法推理像2−2 × $i \times j + j2$这样的数论结构，并决定其输出总是≥0。因此，UOC引入的DNN运算符的模糊序列不会被视为“死码”并容易消除

Per-Input Obscure Condition (PIOC).

PIOC的实现分两个步骤：

1.分析阶段：将种子模型部署到微软的onnxruntime上进行分析以获取其中的数据。
当模型的输入为i时，在m上随便随机选取一个结点n并在数据中找到它的输出O。

2.蜕变阶段：我们引入一个产生恒定输出O的DNN节点（也就是常量结点），然后再构造一个减法的结点，该结点的输入就是结点n的输出以及构造的常量结点。显而易见，最终的输出会是0.

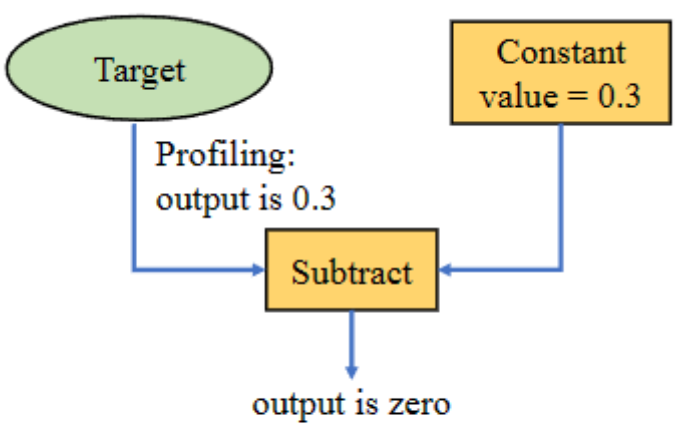


Fig. 6. View of PIOC.

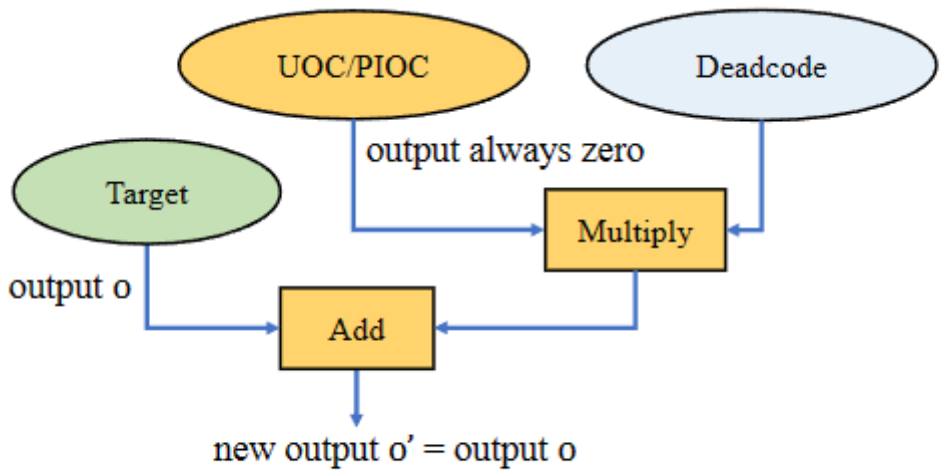
在这里说明，上面使用的函数并不是唯一的，只要能够达到效果都就行

例如可以把完全平方函数的换成下面的函数
ReLU(sin x3 − ln(sin2 x / （2 + cos x） + cos3 x + 10)) 这里是sinx的平方和cosx的三次方
which, though highly obscure, will always output zero. Note that the maximum value of sin x3 is 1, and sin2 x/(2 + cos x) cannot be negative since its numerator is non-negative and its denominator is always positive. cos3 x is always greater or equal to −1. Therefore, the value of $ln(\cdot)$ is no less than ln (0 − 1 + 10) = 2.30. Thus, the input of ReLU is negative,

论文蜕变测试的进行：

1.选取DNN模型种子：主要选取常见的3个图像分类模型

2.Mutating DNN Computation Graph with Two MRs ：



ig. 4. Mutating DNN computation graph with UOC and PIOC.

UOC和PIOC方案都在计算图上引入了许多额外的DNN算子和边，而不影响预测输出，而垃圾结点的引入可以使蜕变的过程更加的复杂但是仍然不会影响输出。因为垃圾结点的输出会与UOC和PIOC的输出相乘，最终还是得到0.

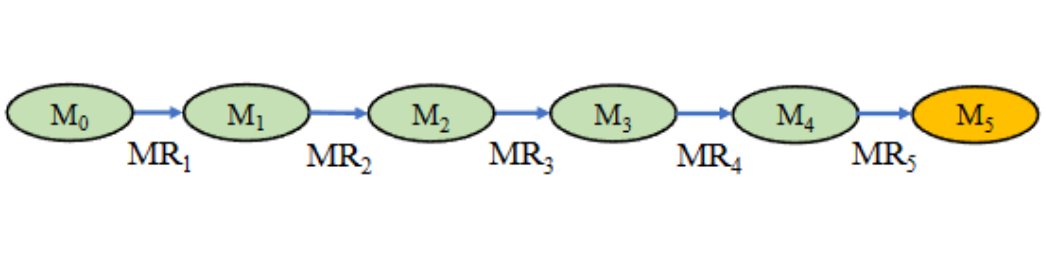
当然只是一次蜕变显然不能满足要求，因此会进行多次的迭代蜕变：
从种子模型M开始，我们迭代变异产生一系列的迭代模型m1,m2.....m。当前迭代模型都来源前一个模型通过选择是UOC还是PIOC的方式进行迭代的。 每个模型都具有相同的功能。m1--->m2--->>>>m.
当然在进行蜕变之前我们都要选择一个目标结点进行突变，也就是Target结点
对于当前实现，我们随机选择一个位置进行变异。

3.Reducing Error-Triggering Models

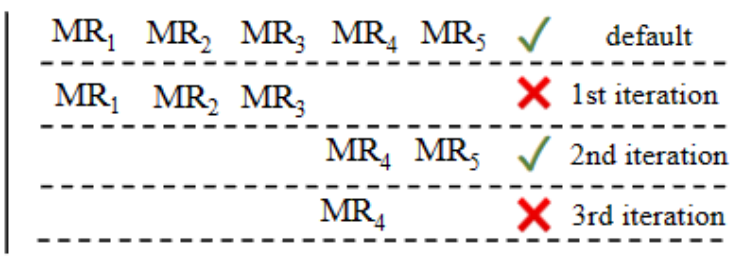
蜕变之后的蜕变模型m有不计其数的操作符，使用这种模型去调式编译器很不友好。

简化方法：通过减少其计算图中不需要重现bug的节点，从而最小化错误触发DNN模型。

方法如下：以M为种子模型进行蜕变迭代，直到m5的时候发生错误触发提示，为了找到错误的模型，可以采用分而治之的方法，（有点类似于二分查找）,如图所示



(a) Iteratively mutating a model until triggering errors at M_5.



(b) Delta-debugging. ✓ denotes still trigger errors while ✗ denotes not.

m1,m2,,m3为一组于M一起测试发现没问题，m4,m5一组有问题....直到找到触发错误提示的模型m5
也就是说，如果bug是由左半部分（MR1、MR2、MR3）和右半部分（MR4、MR5）的一些突变的组合触发的，MT-DLComp将在保留左半部分（MR1、MR2、MR3）的同时在右半部分（MR4、MR5）内启动进一步的搜索，并且类似地，在保留右半部分（MR4、MR5）的同时在左半部分内进行搜索。

然而值得一说的是：某些蜕变模型插入的运算符可能与先前的蜕变模型的输出有关，也就是说，某些突变（MR和MR）具有依赖性，应该一起使用。因此，在增量调试迭代期间，当应用突变MR时，我们包括其所有依赖的MR，并且这种依赖关系可能不可避免地包括许多MR。

4.人工检查调试

由于本实验的目的是检测编译器中的错误，因此有了最小化的错误蜕变模型（m），怎么找到编译器的错误很重要。

（1.）本地化错误的DNN运算操作：
将m编译得到的可执行程序e在反编译器中反编译出来并收集其中的所有汇编功能的信息。然后，通过读取汇编函数名称，我们可以将每个汇编函数映射到DNN运算符。之后就执行e并查看每个汇编函数的输入是否正确。

（2.）本地化编译期间导致错误的IR语句：
当错误的操作符O已经被发现并本地化时，我们就可以在编译时到处并检查编译器发出的IR程序，以开发神经网络交换的格式（ONNX）对比IR程序和m，尤其是操作运算O以及相近的操作，将这两者的连通性和计算图上的相应的结点和边进行比较。
我们的目标是决定IR中的某个操作符是否连接到不正确的节点，这样就能识别出IR过程中的编译器错误。

（3.）本地化编译器源代码的错误模块：
当获取了错误的IR语句并本地化后，我们就可以对编译器的源代码进行迭代测试，找出能够有助于优化和编译这个错误IR语句的模块并将其本地化。